Nicht für den Prüfling bestimmt

I Erläuterungen

Voraussetzungen gemäß KCBG und Abiturerlassen BG jeweils in der für den Abiturjahrgang geltenden Fassung

Standardbezug
Die nachfolgend ausgewiesenen Kompetenzbereiche sind für die Bearbeitung der jeweiligen Aufgabe besonders bedeutsam. Darüber hinaus können weitere, hier nicht explizit ausgewiesene Kompetenzen für die Bearbeitung der Aufgabe nachrangig bedeutsam sein, zumal die Kompetenzen in engem Bezug zueinander stehen. Die Operationalisierung des Bezugs zu den Kompetenzbereichen des Standardbezugs erfolgt in Abschnitt II.

Aufgabe	Kompetenzen				
	K1	**K2**	**K3**	**K4**	**K5**
1.1	X	X			
1.2		X	X		
1.3				X	
1.4			X		
1.5	X				
1.6				X	
1.7				X	
2.1	X	X			
2.2			X		
2.3			X		X
2.4.1			X		
2.4.2			X		
2.4.3				X	
2.4.4				X	

Inhaltlicher Bezug
Die nachfolgend ausgewiesenen Themenfelder sind die wesentliche inhaltliche Grundlage für die vorliegenden Aufgaben. Darüber hinaus können weitere, hier nicht explizit ausgewiesene Themenfelder für die Bearbeitung nachrangig bedeutsam sein.
Q1: Objektorientierte Softwareentwicklung
Q2: Datenbanksysteme
verbindliche Themenfelder: Objektorientierte Modellierung (Q1.1), Implementierung von Klassen und Assoziationen (Q1.2), Datenstrukturen (Q1.4), Konzeptionelle und logische Modellierung einer Datenbank (Q2.1), Datenabfrage und Datenmanipulation mit SQL (Q2.2)

II Lösungshinweise

In den nachfolgenden Lösungshinweisen sind alle wesentlichen Gesichtspunkte, die bei der Bearbeitung der einzelnen Aufgaben zu berücksichtigen sind, konkret genannt und diejenigen Lösungswege aufgezeigt, welche die Prüflinge erfahrungsgemäß einschlagen werden. Selbstverständlich sind jedoch Lösungswege, die von den vorgegebenen abweichen, aber als gleichwertig betrachtet werden können, ebenso zu akzeptieren.

Nicht für den Prüfling bestimmt

Aufg.	erwartete Leistungen	BE		
		I	II	III
1.1	nennen, erläutern Einfache Assoziationen: Zwischen den Klassen `Container` und `Hafen` besteht eine einfache unidirektionale Assoziation. Der Container kennt den Hafen (Rollenname `ziel`), zu dem er transportiert werden soll. Der Hafen kennt den Container nicht, weil diese Navigationsrichtung explizit ausgeschlossen ist. Die Multiplizität bei der Klasse `Hafen` ist eins. Es handelt sich um eine Muss-Beziehung. Die Assoziation wird durch eine Referenzvariable des jeweiligen Zieltyps implementiert. Die Klasse `Container` erhält eine Referenzvariable vom Typ `Hafen`. Zwischen den Klassen `Blocklager` und `Container` besteht eine unidirektionale Eins-zu-viele-Beziehung. Im Blocklager lagern beliebig viele Container. Assoziationen mit der Multiplizität viele werden mit Sammlungen von Referenzvariablen des Typs der Zielkasse umgesetzt. Das Blocklager besitzt eine Sammlung von Referenzvariablen des Typs `Container`. Außerdem enthält das Diagramm sog. „ist-Teil-von"-Beziehungen. Aggregation: Ein Hafen ist Bestandteil einer Schiffsroute. Die Route beginnt mit dem ersten Hafen. Diese Beziehung zwischen einem Aggregat (Route) und seinen Teilen (Hafen) wird durch eine Aggregation zwischen den Klassen `Route` und `Hafen` abgebildet. Die transparente Raute markiert die Aggregatklasse. Komposition: Der Frachtraum des Containerschiffs (Aggregat) besteht aus vielen Bays. Diese sind untrennbar mit dem Containerschiff verbunden. Eine Bay besteht aus vielen nebeneinander angeordneten Stacks. Dieser Sachverhalt wird durch die Komposition zwischen den Klassen `Containerschiff` und `Bay` sowie `Bay` und `Stack` abgebildet. Kompositionen werden durch ausgefüllte Rauten auf der Seite der Aggregatklasse gekennzeichnet. Beide Kompositionen beschreiben 1-zu-viele-Beziehungen. Aggregationen werden in derselben Weise implementiert wie einfache Assoziationen. Bei Kompositionen werden die Teile in der Regel durch das Aggregat erzeugt.			
	nennen	3		
	erläutern		3	
1.2	entwickeln, zeichnen : ContainerSchiff anzahlBays = 20 name = Flying Dutchman : Bay bayNr = 2 : Stack rowNr = 10 bayNr = 2 tiers = 13 : Container bicCode = ABMU3301605 stauCode = 021002 : Container bicCode = QSHI8339397 stauCode = 021004 : Container bicCode = TFGU1849355 stauCode = 021006 : Hafen locode = NLROT ort = Rotterdam			
	entwickeln		5	
	zeichnen	2		

Nicht für den Prüfling bestimmt

Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.3	überführen, implementieren			

```
public class Stack {
  private int rowNr;
  private int bayNr;
  private int tiers;
  private Container top = null;

  public Stack(int rowNr, int bayNr, int tiers) {
    this.rowNr = rowNr;
    this.bayNr = bayNr;
    this.tiers = tiers;
  }

  private int nextTierNr() {
    return (size() + 1) * 2;
  }

  public boolean push(Container container) {
    boolean isPushed = false;
    if (!isClosed()) {
      container.setNext(top);
      top = container;
      // Führende Nullen bei einstelligen Nummern ergänzen
      String stauCode = "";
      stauCode += bayNr < 10 ? "0" + bayNr : bayNr;
      stauCode += rowNr < 10 ? "0" + rowNr : rowNr;
      int tierNr = nextTierNr();
      stauCode += tierNr < 10 ? "0" + tierNr : tierNr;
      container.setStauCode(stauCode);
      isPushed = true;
    }
    return isPushed;
  }

  public Container pop() {
    if (!this.isEmpty()) {
       Container temp = this.top;
      this.top = top.getNext();
      temp.setNext(null);
      return temp;
    }
    return null;
  }

  public boolean isEmpty() {
    return top == null;
  }

  public boolean isClosed() {
    return size() >= tiers;
  }

  public String findeStauCode(String containerID) {
    String stauCode = null;
    Container current = top;
    while (current != null && stauCode == null) {
```

Nicht für den Prüfling bestimmt

Aufg.	erwartete Leistungen	BE I	BE II	BE III

	<pre> if (current.getId().equals(containerID)) {
	        stauCode = current.getStauCode();
	      }
	      current = current.getNext();
	    }
	    return stauCode;
	  }
	
	  public int size() {
	    int size = 0;
	    Container current = top;
	    while (current != null) {
	      current = current.getNext();
	      size++;
	    }
	    return size;
	  }
	}</pre>			

	überführen	4		
	implementieren		5	7

Aufg.	**erwartete Leistungen**	**BE** I	**BE** II	**BE** III
1.4	entwickeln, zeichnen **berechnePruefziffer(owner: String, serienNr: String)** owner und serienNr zu temporärer ID verketten Array werte für 10 Zahlen anlegen für jedes Zeichen aus temporärer ID Ist das Zeichen ein Buchstabe? J: weise dem Buchstaben einen äquivalenten Zahlenwert zu und speichere diesen an der Position des Zeichens in werte N: speichere die Ziffer an der Position des Zeichens in werte summe := 0 zähle i von 0 bis 9, Schrittweite 1 summe := summe + werte[i] * 2^i x := summe / 11 * 11 pruefziffer := summe – x Ist die pruefziffer gleich 10? J: pruefziffer := 0 N: ∅ return pruefziffer			
	entwickeln		3	3
	zeichnen	2		

Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.5	angeben Ein Anwendungsfalldiagramm ermöglicht die grafische Beschreibung von wesentlichen Funktionen (Anwendungsfällen) eines Systems auf abstrakter Ebene. Es stellt die Beziehungen zwischen Akteuren und den Anwendungsfällen dar.	1		
	beschreiben – Das System wird durch die Summe aller Anwendungsfälle beschrieben. Diese werden durch die Systemgrenze zusammengefasst. Der Name des Systems steht innerhalb der Systemgrenze (hier „Containerlogistik“). – Ein Akteur modelliert einen Typ oder eine Rolle, die ein externer Benutzer oder ein externes System während der Interaktion mit dem System einnimmt (z.B. „Blocklager“). – Ein Anwendungsfall (z.B. „Ladeliste erstellen“) wird durch eine Ellipse innerhalb der Systemgrenze dargestellt. In der Ellipse steht die Funktionsbeschreibung. – Die Akteure werden über Verbindungslinien (Beziehungen) mit den Anwendungsfällen verbunden. Eine Beziehung zwischen einem Akteur und einem Anwendungsfall besagt, dass der Akteur an diesem beteiligt ist oder diesen auslösen kann. – Eine include-Beziehung modelliert die unbedingte Einbindung der Funktionalität eines Anwendungsfalls in einen anderen Anwendungsfall (muss-Beziehung). Hier: Der Anwendungsfall „Container laden“ bindet die Funktionalität des Anwendungsfalls „Staucode vergeben“ zwingend ein. – Eine extend-Beziehung modelliert die bedingte Einbindung der Funktionalität eines Anwendungsfalls in einen anderen Anwendungsfall (kann-Beziehung). Hier: Der Anwendungsfall „Schwerlastsicherungen montieren“ erweitert den Anwendungsfall „Container laden“ und beschreibt, dass Schwerlastsicherungen zu montieren sind, falls ein Schwerlastcontainer verladen wird.	2	2	
1.6	implementieren (Code siehe unten)	2	3	3

```java
public List<Container> erstelleStauliste(Route route) {
  List<Container> forward = new List<>();
  Hafen current = route.getFirst();
  while (current != null) {
    for (Container c : container) {
      if (c.getZiel().equals(current) &&
          !forward.contains(c)) {
        forward.add(c);
      }
    }
    current = current.getNext();
  }
  List<Container> reverse = new List<>();
  for (Container c : forward) {
    reverse.add(0,c);
  }
  return reverse;
}
```

Aufg.	erwartete Leistungen	BE		
		I	II	III

1.7	implementieren
	`public List<Container> staueContainer(`
	`                  List<Container> stauListe, Bay bay) {`
	`  int start;`
	`  for (start = 0; bays[start] != bay; start++);`
	`  int j = start;`
	`  do {`
	`    Stack[] stacks = bays[j].getStacks();`
	`    for (int i = 0; !stauListe.isEmpty()`
	`                     && i < stacks.length; i++) {`
	`      Stack stack = stacks[i];`
	`      while (!stauListe.isEmpty() && !stack.isClosed()) {`
	`        Container c = stauListe.get(0);`
	`        if (stack.push(c)) {`
	`          stauListe.remove(c);`
	`        }`
	`      }`
	`    }`
	
	`    if (j < bays.length - 1) {`
	`        j++;`
	`    }`
	`    else {`
	`        j = 0;`
	`    }`
	`  } while(!stauListe.isEmpty() && j != start);`
	`  return stauListe;`
	`}`
		2	3	5

	Summe 60	**18**	**24**	**18**

Aufg.	erwartete Leistungen	BE		
		I	II	III
2.1	beschreiben Die Tabelle ist nicht normalisiert. Die Werte in mehreren Spalten sind nicht atomar (Spalten Reederei, Route, Terminal). Es treten Wiederholungsgruppen auf wie z.B. die Routen, die teilweise dieselben Häfen enthalten. Das Suchen nach Zielhäfen wird durch die Gruppenbildung zudem erschwert. Außerdem treten viele Redundanzen auf, die zu Inkonsistenzen führen können. So stimmen beispielsweise die Adressenangaben des Terminal Altenwerder nicht überein. Außerdem enthält die Tabelle unterschiedliche Schreibweisen von Schiffs- und Reedereinamen.	2		
	erläutern Änderungsanomalie: Falls die Reederei beispielsweise ihren Firmensitz verlegt, muss die Änderung in allen Feldern der Tabelle durchgeführt werden. Wird ein Eintrag übersehen, kommt es zu einer Änderungsanomalie. Löschanomalie: Wird ein Containerschiff aus der Tabelle gelöscht (z.B. Seamaster), würden auch die Informationen über die Reederei und das Terminal aus der dargestellten Tabelle entfernt. Einfügeanomalie: Wenn eine neue Reederei in die Tabelle aufgenommen wird, bleiben sehr viele Felder leer bzw. enthalten null-Werte.		3	

Aufg.	erwartete Leistungen	BE I	BE II	BE III
2.2	entwickeln, zeichnen Reederei (rdid, name, sitz) [1, 1] – hat – [0, n] Containerschiff (csid, name, kapazitaet) Containerschiff [0, m] – faehrt – [0, n] Route (rid) Containerschiff [0, n] – liegt (beginn, ende) – [0, m] Liegeplatz (lpid, bezeichnung) Route [0, n] – hat (position) – [1,m] Hafen (hid, name) Liegeplatz [1, n] – hat – [1, 1] Terminal (tid, name, adresse) Hafen [1, 1] – hat – [1, n] Terminal entwickeln zeichnen	 3	4	2

Aufg.	**erwartete Leistungen**	**BE**		
		I	**II**	**III**

2.3	überführen
	`Reederei(rdid, name, sitz)`
	`Containerschiff(csid, name, kapazitaet, rdid#)`
	`Hafen(hid, name)`
	`Terminal(tid, name, strasse, hausnr, plz, ort, hid#)`
	`Liegeplatz(lpid, bezeichnung, tid#)`
	`Liegt(lid, beginn, ende, lpid#, csid#)`
	`Route(rid)`
	`HatHafen(rid#, hid#, position)`
	`Faehrt(csid#, rid#)`
	
	begründen
	Alle Attribute der Relationen sind zu atomisieren (1. NF). Name und Sitz der Reederei werden getrennt, die Adresse der Terminals wird aufgelöst. Die Relationen `Reederei`, `Hafen`, `Terminal`, `Liegeplatz` und `Liegt` erhalten synthetische PKs. Damit sind alle Nichtschlüsselattribute voll funktional vom PK abhängig (2. NF). Transitive Abhängigkeiten werden aufgelöst, so dass keine Redundanzen mehr auftreten (3. NF). Die Relationen `HatHafen`, `Faehrt` und `Liegt` sind Zwischentabellen, die n:m-Beziehungen abbilden.
	
	überführen
	begründen
		2	3
			1
				
				2

2.4.1	angeben
	`SELECT cs.name, cs.csid, cs.kapazitaet`
	`FROM Containerschiff cs JOIN Reederei r USING(rdid)`
	`WHERE r.name = 'Marsek';`
		3		

2.4.2	formulieren
	`SELECT cs.Name`
	`FROM Containerschiff cs`
	`JOIN Liegt l USING(csid)`
	`JOIN Liegeplatz lp USING(lpid)`
	`JOIN Terminal t USING(tid)`
	`WHERE t.name = 'Altenwerder'`
	`AND l.beginn BETWEEN '2022-03-20 00:00:00' AND`
	`'2022-03-25 23:59:59'`
	`AND l.ende BETWEEN '2022-03-20 00:00:00' AND`
	`'2022-03-25 23:59:59';`
			4	

2.4.3	implementieren
	`SELECT t.name, lp.bezeichnung FROM Terminal t`
	`JOIN Liegeplatz lp USING(tid)`
	`WHERE t.hid = (SELECT hid FROM Hafen WHERE name = 'Hamburg')`
	`AND lp.lpid NOT IN`
	`(SELECT lp.lpid FROM Liegeplatz lp`
	`JOIN liegt l USING(lipd)`
	`WHERE`
	`(l.beginn BETWEEN '2022-03-20 10:00:00' AND`
	`'2022-03-20 22:00:00'`
	`OR l.ende BETWEEN '2022-03-20 10:00:00' AND`
	`'2022-03-20 22:00:00')`
	`OR`
	`(l.beginn < '2022-03-20 10:00:00' AND`
	`l.ende > '2022-03-20   22:00:00'))`
	`ORDER BY t.name, lp.bezeichnung`
			2	4

Aufg.	erwartete Leistungen	BE		
		I	II	III

2.4.4	entwickeln
	`SELECT t.name AS Terminal, lp.lpid, lp.bezeichnung AS Liegeplatz,`
	`SUM(TIME_TO_SEC(TIMEDIFF(l.ende, l.beginn))/3600) AS belegt,`
	`SUM(TIME_TO_SEC(TIMEDIFF(l.ende, l.beginn))/3600)/(365*24) AS Auslastung`
	`FROM Terminal t`
	`JOIN Liegeplatz lp USING(tid)`
	`JOIN Liegt l USING(lpid)`
	`WHERE t.hid = 'NLROT'`
	`AND YEAR(l.beginn) = 2021 AND YEAR(l.ende) = 2021`
	`GROUP BY lp.lpid`
	`ORDER BY Auslastung DESC;`
			2	3

	Summe 40	**10**	**19**	**11**

Nicht für den Prüfling bestimmt

III Bewertung und Beurteilung

Die Bewertung und Beurteilung erfolgt unter Beachtung der nachfolgenden Vorgaben nach § 33 der Oberstufen- und Abiturverordnung (OAVO) in der jeweils geltenden Fassung. Bei der Bewertung und Beurteilung der sprachlichen Richtigkeit in der deutschen Sprache sind die Bestimmungen des § 9 Abs. 12 Satz 3 OAVO in Verbindung mit Anlage 9b anzuwenden.

Bei der Bewertung und Beurteilung der Übersetzungsleistung in den Fächern Latein und Altgriechisch sind die Bestimmungen des § 9 Abs. 14 OAVO in Verbindung mit Anlage 9c anzuwenden.

Der Fehlerindex ist nach Anlage 9b zu § 9 Abs. 12 OAVO zu berechnen. Für die Ermittlung der Punkte nach Anlage 9a zu § 9 Abs. 12 OAVO sowie Anlage 9c zu § 9 Abs. 14 OAVO wird jeweils der ganzzahlige nicht gerundete Prozentsatz bzw. Fehlerindex zugrunde gelegt.

Für die Bewertung in den modernen Fremdsprachen ist der „Erlass zur Bewertung und Beurteilung von schriftlichen Arbeiten in allen Grund- und Leistungskursen der neu beginnenden und fortgeführten modernen Fremdsprachen in der gymnasialen Oberstufe, dem beruflichen Gymnasium, dem Abendgymnasium und dem Hessenkolleg“ vom 7. August 2020 (ABl. S. 519) zugrunde zu legen. Demnach erfolgt die Bewertung und Beurteilung mit der Maßgabe, dass lediglich bei der Ermittlung des Prüfungsergebnisses (Note) aus Prüfungsteil 1 und 2 gerundet wird.

Darüber hinaus sind die Vorgaben der Erlasse „Hinweise zur Vorbereitung auf die schriftlichen Abiturprüfungen (Abiturerlass)“ und „Durchführungsbestimmungen zum Landesabitur“ in der für den Abiturjahrgang geltenden Fassung zu beachten.

Als Kriterien für die Bewertung und Beurteilung dienen unter Beachtung der Zielsetzung der gymnasialen Oberstufe nach § 1 Abs. 2 OAVO neben dem Inhaltlichen auch die in den Kerncurricula genannten überfachlichen Kompetenzen, insbesondere die Sprachkompetenz und Wissenschaftspropädeutik; dies zeigt sich u.a. in qualitativen Merkmalen wie Strukturierung, Differenziertheit, (fach-)sprachlicher Gestaltung und Schlüssigkeit der Argumentation.

Im Fach Praktische Informatik besteht die Prüfungsleistung aus der Bearbeitung eines Vorschlags, wofür insgesamt maximal 100 BE vergeben werden können. Ein Prüfungsergebnis von **5 Punkten (ausreichend)** setzt voraus, dass mindestens 45% der zu vergebenden BE erreicht werden. Ein Prüfungsergebnis von **11 Punkten (gut)** setzt voraus, dass mindestens 75% der zu vergebenden BE erreicht werden.

Gewichtung der Aufgaben und Zuordnung der Bewertungseinheiten zu den Anforderungsbereichen

Aufgabe	Bewertungseinheiten in den Anforderungsbereichen			Summe
	AFB I	AFB II	AFB III	
1	18	24	18	**60**
2	10	19	11	**40**
Summe	**28**	**43**	**29**	**100**

Die auf die Anforderungsbereiche verteilten Bewertungseinheiten innerhalb der Aufgaben sind als Richtwerte zu verstehen.